Bulletin of Natural Sciences Research
Vol. 15, No. 1, 2025, pp. 72-77.

DOI: https://doi.org/10.5937/bnsr15-58114
Original Scientific Paper

AN APPLICATION OF RESIDUE NUMBER SYSTEM ARITHMETICS TO SECURE HASH FUNCTIONS DESIGN

MILIJA PAVLOVIĆ[1,⋆], STEFAN PANIĆ[1], BORIS DAMJANOVIĆ[2], NEGOVAN STAMENKOVIĆ[1]

[1]Faculty of Natural Sciences and Mathematics, University of Priština in Kosovska Mitrovica, Kosovska Mitrovica, Serbia
[2]Faculty of Information Technologies and Engineering, Union - Nikola Tesla University, Belgrade, Serbia

ABSTRACT

This paper presents a cryptographic hash function based on the Residue Number System (RNS), designed to enhance security and computational efficiency. The function leverages the parallelism and modular properties of RNS to achieve high-speed processing while maintaining strong diffusion and resistance to various cryptanalytic attacks. Experimental results confirm that the proposed function exhibits a pronounced Avalanche effect, ensuring that minor changes in the input result in significant alterations in the hash output. Additionally, statistical analysis using the ENT test demonstrates a high level of entropy and uniform distribution of hash values, reinforcing the function's unpredictability—an essential characteristic for cryptographic security. The proposed hash function is suitable for applications in digital signatures, data integrity verification, and authentication systems, offering advantages in environments requiring high computational efficiency.

Keywords: Cryptographic hash function, Residue number system, Avalanche effect, Diffusion, Entropy, Parallelism, Modular arithmetic.

INTRODUCTION

The main contribution of this paper is the proposal of RNS to parallelizing hash function operations, to provide speed boost while preserving cryptographic security. The Residue Number System (RNS) is an unconventional numerical system that achieves high efficiency in data processing, thanks to arithmetic based on modular operations (Stamenković, 2019). In recent decades, the application of RNS in digital signal processing, cryptography, and many other areas of modern computing has attracted significant attention (Ananda Mohan, 2016; Omondi & Premkumar, 2007; Szabo & Tanaka, 1967).

It ensures the efficiency of arithmetic operations due to the high degree of parallelization, making it particularly interesting for researchers working on computationally intensive applications. This numerical system is based on converting a large number into several smaller numbers, which are residues obtained as a result of dividing the given number by moduli, where the moduli are pairwise coprime integers (Isupov, 2021).

One of the most significant advantages of RNS lies in the fact that the residues are mutually independent. This means that addition, subtraction, and multiplication operations are reduced to independent operations with shorter residue values, eliminating the need for carry propagation between them and avoiding computationally expensive operations with long bit-lengths (Ananda Mohan, 2016).

When discussing the application of RNS, it is most often associated with Montgomery modular multiplication (Montgomery, 1985). Since the mid-2000s, numerous studies have explored RNS-based Montgomery multiplication and the optimization of cryptosystem design. RNS arithmetic was first utilized in elliptic curve cryptography (ECC) in , and it has also been employed in RSA cryptography (Bajard & Imbert, 2004; Gandino et al., 2011, 2012).

Beyond its applications in asymmetric cryptography, RNS has also been studied in the context of hash functions, where its properties are leveraged to enhance efficiency and security (Naseem et al., 2013). Research shows that using RNS in hashing can improve parallel processing and reduce resource consumption, which is particularly important for hash functions with large input data lengths. Furthermore, certain RNS-based hash function constructions demonstrate increased resistance to specific types of cryptanalytic attacks, making them promising candidates for modern security systems.

HASH FUNCTIONS IN THE CONTEXT OF RNS

Hash functions are a fundamental element of modern cryptographic systems, providing data compression into a fixed size and ensuring information integrity (Rogaway & Shrimpton, 2004). Traditional hashing principles are based on a deterministic transformation of input data into a hash value, which is cryptographically strong if it satisfies the following properties:

- pre-image resistance,
- second pre-image resistance,
- collision resistance.

⋆ Corresponding author: milija.pavlovic@pr.ac.rs

These characteristics enable hash functions to serve as the foundation for digital signatures, data integrity verification mechanisms, and password storage algorithms (Ambedkar, 2025).

Although traditional hashing methods offer solid security guarantees, their intensive computational complexity can be a limitation in resource-constrained environments, such as IoT devices or embedded systems (Stevens et al., 2017). In this context, an increasing number of studies are exploring the application of unconventional computing techniques, such as the Residue Number System (RNS), which allows carry-free arithmetic, thereby improving processing speed and reducing resource consumption (Köroğlu, 2025).

By applying RNS in the design of hash functions, algorithms can achieve better parallelization of operations, leading to faster hashing without compromising security properties. Moreover, the resistance of RNS to certain cryptanalytic techniques, such as zero-value attacks, makes it promising for the development of new generations of hash functions. Research shows that RNS-based hash functions can be more efficient in high-throughput systems, such as blockchain technologies and data protection in large-scale information infrastructures.

RNS HASH FUNCTION

Let M be a given message of length N bits. The message is segmented into blocks of 128 bits as follows:

1. If N is divisible by 128, the message is divided into $t = \frac{N}{128}$ blocks without any padding.
2. If N is not divisible by 128, the message is divided into $t = \lceil N/128 \rceil$ blocks, where the last block, if shorter than 128 bits, is left-padded with zeros to reach the full length of 128 bits.

Formally, let

$$t = \lceil N/128 \rceil. \tag{1}$$

be the number of blocks. The message M is then represented as a sequence of blocks:

$$M = B_1 \| B_2 \| \dots \| B_t. \tag{2}$$

where for each i, where $1 \le i < t$, we have

$$B_i = M[(i-1) \cdot 128 : i \cdot 128]. \tag{3}$$

while the last block B_t is formed as

$$B_t = 0^{128-(N \mod 128)} \| M[(t-1) \cdot 128 : N]. \tag{4}$$

where 0^k denotes a sequence of k zeros, and $\|$ denotes concatenation.

Each block B_i is first incremented by a constant value of 255, resulting in a transformed block:

$$B'_i = B_i + 255. \tag{5}$$

After this transformation, a set of moduli $\{x_1, x_2, \dots, x_8\}$ is used, with the following modulus values:

$$\begin{aligned} &x_1 = 173, \quad x_2 = 179, \quad x_3 = 181, \quad x_4 = 191, \\ &x_5 = 193, \quad x_6 = 197, \quad x_7 = 199, \quad x_8 = 211, \\ &x_9 = 223, \quad x_{10} = 227, \quad x_{11} = 229, \quad x_{12} = 233, \\ &x_{13} = 239, \quad x_{14} = 241, \quad x_{15} = 247, \quad x_{16} = 251. \end{aligned} \tag{6}$$

The selection of moduli was made with the constraint that the length of the residues must not exceed 8 bits, which implies that the moduli themselves must not be greater than 255 — the largest number that can be represented in an 8-bit system. For this reason, the largest possible prime numbers less than 255 were selected, in order to cover as many distinct residues as possible, reduce the probability of collisions, and preserve the relative primality among the moduli, which is a key property for the correct functioning of the Residue Number System (RNS). The only exception in the set is the number 247, which is not prime, but is relatively prime to all other selected moduli, allowing it to be used without violating the fundamental principles of RNS.

For each modulus x_j, the remainder of the division of the transformed block B'_i is computed, along with the integer quotient:

$$a_{i,j} = B'_i \mod x_j, \quad A_{i,j} = \lfloor B'_i / x_j \rfloor, \quad j \in \{1, 2, \dots, 16\}. \tag{7}$$

These parameters enable further numerical processing within the Residue Number System (RNS) framework, improving computational efficiency and security in the applied algorithm.

After computing the initial set of values $\{a_{i,j}, A_{i,j}\}$, the processing continues through an iterative update of remainders and quotients, enabling more efficient data manipulation within the Residue Number System (RNS) framework. This procedure consists of multiple stages in which previously computed values are used to generate new parameters.

For each modulus x_j, the new remainder is calculated by taking the sum of the previous quotient and remainder modulo x_j:

$$b_{i,j} = (A_{i,j} + a_{i,j}) \mod x_j, \quad j \in \{1, 2, \dots, 16\}. \tag{8}$$

Simultaneously, the new quotient is determined as the integer part of the same expression divided by x_j:

$$B_{i,j} = \left\lfloor \frac{A_{i,j} + a_{i,j}}{x_j} \right\rfloor, \quad j \in \{1, 2, \dots, 16\}. \tag{9}$$

After completing this phase, the obtained values $B_{i,j}$ and $b_{i,j}$ serve as input for the next iteration, where a new set of remainders and quotients is computed using the same principle:

$$c_{i,j} = (B_{i,j} + b_{i,j}) \mod x_j, \quad C_{i,j} = \left\lfloor \frac{B_{i,j} + b_{i,j}}{x_j} \right\rfloor. \tag{10}$$

This process is repeated iteratively across multiple steps, where each subsequent phase utilizes the previously computed values to generate new ones:

$$\begin{aligned} &\{d_{i,j}, D_{i,j}\}, \quad \{e_{i,j}, E_{i,j}\}, \quad \{f_{i,j}, F_{i,j}\}, \quad \{g_{i,j}, G_{i,j}\}, \\ &\{h_{i,j}, H_{i,j}\}, \quad \{i_{i,j}, I_{i,j}\}, \quad \{j_{i,j}, J_{i,j}\}, \quad \{k_{i,j}, K_{i,j}\}, \\ &\{l_{i,j}, L_{i,j}\}, \quad \{m_{i,j}, M_{i,j}\}, \quad \{n_{i,j}, N_{i,j}\}, \quad \{o_{i,j}, O_{i,j}\}, \\ &\{p_{i,j}, P_{i,j}\}. \end{aligned} \tag{11}$$

By iteratively repeating this procedure, data is successively transformed through multiple stages, enabling optimized processing within the RNS framework. This method reduces computational errors, enhances efficiency, and increases resistance to attacks in applications where the Residue Number System is utilized.

For each iterative step k, where $k \geq 1$, the values of remainders and quotients are updated according to the following relations:

$$x_{i,j}^{(k)} = (X_{i,j}^{(k-1)} + x_{i,j}^{(k-1)}) \mod x_j, \quad X_{i,j}^{(k)} = \left\lfloor \frac{X_{i,j}^{(k-1)} + x_{i,j}^{(k-1)}}{x_j} \right\rfloor. \tag{12}$$

The initial values are defined as $x_{i,j}^{(0)} = a_{i,j}$ and $X_{i,j}^{(0)} = A_{i,j}$. This process continues until the final step, where the resulting values $\{p_{i,j}, P_{i,j}\}$ are obtained for each block B_i.

The computation of the final hash is performed through the following steps:

First, the final remainders are determined using the XOR operation. For each modulus x_j, where $j = 1, \ldots, 16$. and the corresponding values $a_j, b_j, c_j, d_j, e_j, f_j, g_j, h_j, i_j, j_j, k_j, l_j, m_j, n_j, o_j, p_j, P_j$, the final remainder rem_j is defined as:

$$\begin{aligned} \mathrm{rem}_j = a_j \oplus b_j \oplus c_j \oplus d_j \oplus e_j \oplus f_j \oplus g_j \oplus h_j \oplus i_j \oplus j_j \\ \oplus k_j \oplus l_j \oplus m_j \oplus n_j \oplus o_j \oplus p_j \oplus P_j. \end{aligned} \tag{13}$$

where the symbol $\oplus$ represents the XOR operation.

Next, each result is converted into an 8-bit binary representation. If necessary, leading zeros are added to ensure the appropriate length. The resulting binary representation, denoted as b_rem_j, is computed as:

$$b\_\mathrm{rem}_j = \mathrm{bin}(\mathrm{rem}_j). \tag{14}$$

where leading zeros are added to ensure an exact length of 8 bits.

Finally, the final hash value hesF is formed by concatenating individual binary representations, formally expressed as:

$$\begin{aligned} \mathrm{hesF} = b\_\mathrm{rem}_1 || b\_\mathrm{rem}_2 || b\_\mathrm{rem}_3 || b\_\mathrm{rem}_4 \\ || b\_\mathrm{rem}_5 || b\_\mathrm{rem}_6 || b\_\mathrm{rem}_7 || b\_\mathrm{rem}_8 \\ || b\_\mathrm{rem}_9 || b\_\mathrm{rem}_{10} || b\_\mathrm{rem}_{11} || b\_\mathrm{rem}_{12} \\ || b\_\mathrm{rem}_{13} || b\_\mathrm{rem}_{14} || b\_\mathrm{rem}_{15} || b\_\mathrm{rem}_{16}. \end{aligned} \tag{15}$$

where the symbol || denotes the concatenation operation of binary sequences.

As a result, the final hash hesF is a 128-bit binary sequence, formed by combining all individual binary residues. In Fig. 1. and Fig. 2. we can see a graphical representation of the algorithm.

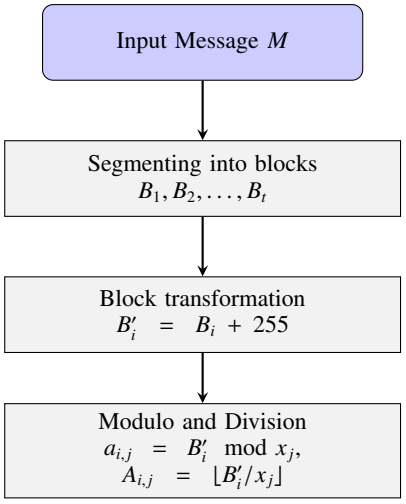

Figure 1. Hashing Process - Part 1: Data Preparation.

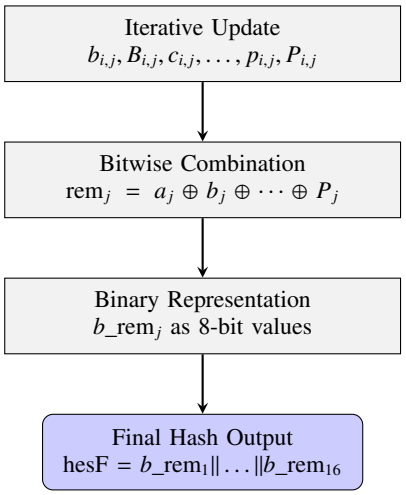

Figure 2. Hashing Process - Part 2: Finalization.

ENT TEST OF RNS HASH FUNCTION

The ENT test is a statistical tool for analyzing data randomness, making it particularly relevant for evaluating cryptographic hash functions. A strong hash function should produce outputs with high entropy, uniform value distribution, and minimal statistical deviations (Walker, 2008).

To determine whether a hash function exhibits the properties of ideal randomness, key parameters of the ENT test, such as entropy, arithmetic mean, Monte Carlo approximation of π, and serial correlation, are used.

Applying the ENT test to hash function outputs allows researchers to identify weaknesses in cryptographic algorithms and verify their suitability for security applications, such as digital signatures, password storage, and data integrity preservation in blockchain technologies.

Entropy (H)

Entropy measures the amount of uncertainty or randomness in a dataset and is defined as:

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i. \tag{16}$$

where p_i is the probability of occurrence of the i-th symbol in the dataset (Cover & Thomas, 1991). The maximum entropy for a completely random binary file is $H = 8$ bits per byte.

Arithmetic Mean ($\bar{x}$)

The arithmetic mean of the bytes in the dataset is calculated as:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i. \tag{17}$$

where x_i are individual byte values, and N is the total number of bytes (Cox & Hinkley, 1980). For perfectly random data, the expected value is 127.5.

Monte Carlo Approximation of π

This test relies on a simulation where random coordinate pairs (x, y) are generated within a unit square. The value of π is estimated using the ratio of points that fall within a circle to the total number of generated points:

$$\pi \approx 4 \times \frac{\text{number of points in the circle}}{\text{total number of points}}. \tag{18}$$

The approximation error indicates deviations in the randomness of the data (Kalos & Whitlock, 2008).

Serial Correlation Coefficient (r)

The serial correlation coefficient measures the correlation between consecutive bytes in a file:

$$r = \frac{\sum (x_i - \bar{x})(x_{i+1} - \bar{x})}{\sum (x_i - \bar{x})^2}. \tag{19}$$

where $\bar{x}$ is the arithmetic mean, and x_i is the value of the i-th byte. Ideally random data should have a value close to 0, while positive or negative values indicate patterns or dependencies between adjacent bytes (Brockwell & Davis, 2002).

Table 1. ENT Test Results for the File `output.bin`.

Parameter	**Obtained Value**	**Expected Value**
Entropy (H)	7.999555	8.000
Arithmetic Mean ($\bar{x}$)	128.8689	127.500
Monte Carlo π Approximation	3.101761	3.141593
Serial Correlation (r)	0.002546	0.000

An ENT test was conducted on a sample of 1,000,000 hash functions, and the obtained results are presented in Tab. 1. The entropy is 7.999555 bits per byte, which is very close to the maximum value of 8 bits per byte, indicating a high level of data unpredictability, as expected for well-generated random numbers.

The sample's compressibility is 0%, which aligns with theoretical expectations, as truly random sequences cannot be efficiently compressed.

The arithmetic mean is 128.8689, which is close to the theoretical value of 127.5, with the minor deviation attributed to statistical fluctuations.

The Monte Carlo estimation of π is 3.101761319, with an error of 1.27% compared to the exact value (3.1415926535). This result is consistent with expectations for random data and confirms their good randomness.

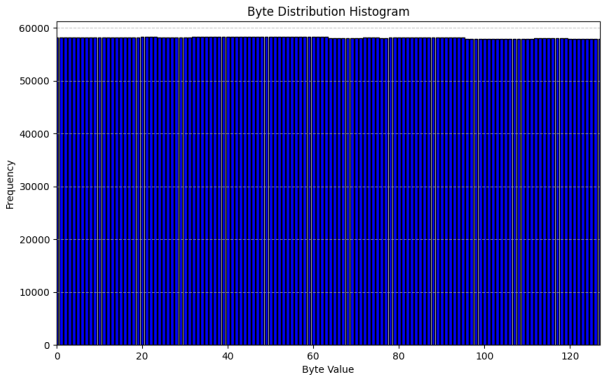

Figure 3. Byte Distribution Histogram (RNS Hash).

The byte distribution histogram shown in Fig. 3. demonstrates a high degree of uniformity, which is a key indicator of the quality of a hash function or a cryptographically secure pseudo-random number generator. The frequencies of all byte values are approximately equal, indicating good diffusion and the absence of bias toward specific values, thereby minimizing the predictability of the output and ensuring strong security. Visual analysis suggests that the data entropy is close to its maximum value, meaning there is no apparent pattern that could be exploited for analysis or attacks. The absence of significant anomalies, such as noticeable peaks or gaps in the byte spectrum, further confirms that the algorithm produces a balanced output without statistical irregularities that could compromise its robustness. The clarity of the visual presentation, with well-defined axes and a grid, facilitates straightforward and precise data interpretation, further emphasizing the reliability and security of the analyzed system.

AVALANCHE EFFECT

The avalanche effect is a crucial property of cryptographic hash functions that ensures minimal changes in input data (e.g., flipping a single bit) result in significant changes to the output hash value. This property is essential for good diffusion, meaning that changing one input bit should affect approximately 50%

of the output bits. The avalanche effect is characterized by two key criteria:

1. Strict Avalanche Criterion (SAC),
2. Bit Independence Criterion.(BIC)(Wang et al., 2012)

For mathematical analysis of these properties, the following definitions are used:

– Input messages: m, n,
– Hamming distance: $D(a, b)$ – number of bit positions where strings a and b differ,
– Hash value: $H(i)$ – output of hash function H for input i,
– Output size: s – fixed length of hash function $H()$ output.

These properties ensure that a hash function properly implements the avalanche effect, which is crucial for its security (Upadhyay et al., 2022).

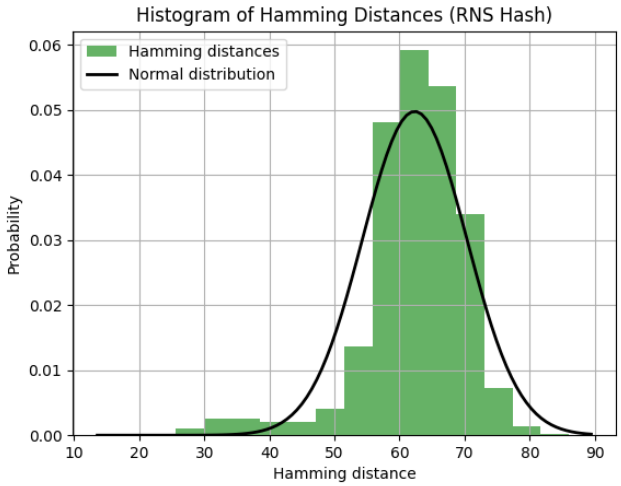

Figure 4. Histogram of Hamming Distances (RNS Hash).

Histogram of Hamming distances shown in Fig. 4. illustrates the distribution of bit changes in the hash output of the RNS hash function. The distribution demonstrates a high level of the avalanche effect, with values concentrated around a mean close to 64, which aligns with the ideal expectation for a 128-bit hash output.

The shape of the distribution follows a normal distribution, confirming that each bit in the hash has approximately an equal probability of changing with minimal input variations. The evident symmetry of the histogram indicates consistent diffusion, while slight variations at the tails do not compromise the overall stability of the algorithm.

The hash function exhibits a high degree of entropy and uniformity in bit changes, suggesting strong robustness for cryptographic applications. The data distribution is well-balanced, without significant anomalies, confirming the reliability of the applied method and its ability to ensure security through a strong avalanche effect.

Table 2. Examples of the Avalanche Effect.

Input Values	RNS Hash Values	Hamming Distances
"Hello World"	D6064077F28E9AC...	-
"hello world"	715FF3A1DBF5CC8F...	65
"hellO worlD"	520C9747994B5DFF...	69
"HELLO WORLD"	F1C4F3C78EC8E321...	65

Tab. 2. presents an analysis of the Avalanche effect by comparing variations of an input string and their corresponding Residue Number System (RNS) hash values. The Hamming distance is used as a metric to quantify the number of bit changes in the hash output when the input undergoes minimal modifications.

The results demonstrate that even small changes in letter case (e.g., lowercase vs. uppercase) lead to significant alterations in the hash values. The measured Hamming distances, ranging from 65 to 69 bits, indicate a strong diffusion property of the RNS-based hashing function, aligning with the expected behavior of a cryptographic hash function. The distribution of distances suggests a high level of randomness and unpredictability, which are desirable characteristics in cryptographic applications.

Furthermore, the table illustrates that the hash function successfully propagates input variations throughout the output space, reinforcing its resilience against pattern detection and potential vulnerabilities. The consistent magnitude of bit changes across different input modifications suggests that the hash function adheres to the strict avalanche criterion (SAC), a fundamental property ensuring security in hashing algorithms.

CONCLUSION

The RNS-based hash function has demonstrated a high level of security characteristics, as confirmed by positive results on the ENT test and a pronounced Avalanche effect. The function's construction, which relies on the Residue Number System (RNS) for parallel and modular data processing, enables efficient and distributed computation, enhancing resistance to certain types of attacks and improving execution speed.

Experimental analysis shows that even the smallest changes in the input data lead to significant alterations in the hash output, indicating strong diffusion and compliance with the Strict Avalanche Criterion (SAC). The results of the ENT test confirm the high entropy of output values, ensuring their uniformity and unpredictability—key properties of a secure hash function.

Due to these characteristics, the RNS-based hash function can be applied in various cryptographic systems, including digital signatures, data integrity verification, and authentication. Its resistance to linear and differential attacks is further strengthened by the modular structure of RNS, making it more difficult to perform reversible analysis or predict outputs based on inputs.

Future research could focus on optimizing the selection of moduli within the RNS to improve the balance of diffusion and processing speed. Additionally, exploring the application of this

hash function in resource-constrained environments could provide significant performance advantages due to its parallelism. Further analyses of resistance to side-channel attacks could also contribute to confirming its security robustness.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support from the Serbian Ministry of Science, Technological Development and Innovation (Contract No. 451-03-65/2024- 03/200124). This work has been partially funded by the University of Pristina in Kosovska Mitrovica, Faculty of Science and Mathematics, Serbia, under the project: IJ-2302 (Optimization of neural network).

REFERENCES

Ambedkar, B. R. 2025, Enhancing the Performance of Cryptographic Hash Function Using 2080 Bits Proposed Secure Hash Algorithm 160, International Journal of Scientific Research in Network Security and Communication, 13 (1), pp. 8–11. https://doi.org/10.26438/ijsrnsc.v13i1.264

Ananda Mohan, P. 2016, Residue Number Systems: Theory and Applications.

Bajard, J. & Imbert, L. 2004, A full RNS implementation of RSA, IEEE Transactions on Computers, 53, pp. 769–774. https://doi.org/10.1109/TC.2004.2

Brockwell, P. J. & Davis, R. A. 2002, Introduction to Time Series and Forecasting, 2nd edn. (Springer), pp. 15–1.

Cover, T. M. & Thomas, J. A. 1991, Elements of Information Theory, Wiley Series in Telecommunications. https://doi.org/10.1002/047174882X

Cox, D. R. & Hinkley, D. V. 1980, Theoretical Statistics, Chapman & Hall, pp. 34–38. https://doi.org/10.1201/9780203758486

Gandino, F., Lamberti, F., Montuschi, P., & Bajard, J. 2011, in Computer Arithmetic (ARITH), 2011 20th IEEE Symposium on, pp. 195–204.

Gandino, F., Lamberti, F., Paravati, G., Bajard, J., & Montuschi, P. 2012, An Algorithmic and Architectural Study on Montgomery Exponentiation in RNS, IEEE Transactions on Computers, 61 (8), pp. 1071–1083. https://doi.org/10.1109/TC.2012.84

Isupov, K. 2021, High-Performance Computation in Residue Number System Using Floating-Point Arithmetic, Computation, 9 (2), 9. https://doi.org/10.3390/computation9020009

Kalos, M. H. & Whitlock, P. A. 2008, Monte Carlo Methods, 2nd edn. (Wiley-VCH), pp. 3–12, poglavlje o historijskoj primjeni za sa matematičkom analizom.

Köroğlu, T. 2025, Hash Functions: Design Paradigms, Security, and Algorithmic Analysis, Advances in Computer Engineering, pp. 109–135. https://doi.org/10.5281/zenodo.14579277

Montgomery, P. 1985, Modular multiplication without trial division, Mathematics of Computation, 16, pp. 519–521. https://doi.org/10.1090/S0025-5718-1985-0777282-X

Naseem, M. T., Qureshi, I. M., Cheema, T. A., & ur Rahman, A. 2013, Hash based Medical Image Authentication and Recovery using Chaos and Residue Number System, Journal of Basic and Applied Scientific Research.

Omondi, A. & Premkumar, B. 2007, Residue Number Systems: Theory and Implementation (London, UK)

Rogaway, P. & Shrimpton, T. 2004, in Lecture Notes in Computer Science, Vol. 3152, Advances in Cryptology – CRYPTO 2004 (Berlin, Heidelberg: Springer), pp. 371–388.

Stamenković, N. 2019, Isomorphic Transformation and Its Application to the Modulo (2^n+1) Channel for RNS Based FIR Filter Design, Bulletin of Natural Sciences Research, 9 (2), pp. 25–32. https://doi.org/10.5937/univtho9-21821

Stevens, M., Bursztein, E., & Karpman, P. e. a. 2017, The First Collision for Full SHA-1, Advances in Cryptology - CRYPTO 2017.

Szabo, N. & Tanaka, R. 1967, Residue Arithmetic and Its Application to Computer Technology (New York, NY, USA)

Upadhyay, D., Gaikwad, N., Zaman, M., & Sampalli, S. 2022, Investigating the Avalanche Effect of Various Cryptographically Secure Hash Functions and Hash-Based Applications, IEEE Access, 10, pp. 112473–112486. https://doi.org/10.1109/ACCESS.2022.3215778

Walker, J. 2008, ENT: A Pseudorandom Number Sequence Test Program, Online manual, the original ENT entropy test documentation with statistical foundations.

Wang, Y., Wong, K.-W., Li, C., & Yang, L. 2012, A novel method to design S-box based on chaotic map and genetic algorithm, Physics Letters A, 376 (6-7), pp. 827–833. https://doi.org/10.1016/j.physleta.2012.01.009